Jeremy Swartwood, UAA Machine Learning, spring 2018, Assignment 3

# Overview

Analysis of the validation results show that most of the kernels did no better than a 50/50 guess coin flip. The best fit kernel for this dataset was the default linear (or poly degree 1, which are essentially the same). I suspect this is because of the two classes (yes/no) of the dataset. Perhaps if we included the unknowns as “0” then a degree 2 poly might have been a better fit. I was a bit surprised that the sigmoid tanh didn’t do so well either.

After performing PCA reduction my feature space went from 768 down to 17.

I reached 85.83% accuracy with the original 768 features, and only lost about 1% accuracy when PCA dropped the features down to 17. In comparison, the perceptron got 94.2% accuracy when I ran 5Million rows through. So it’s a 10% loss to the single perceptron. This took 1155 seconds on a pretty decent computer. SVM was almost instant. So there are tradeoffs to accuracy vs speed.

# SVM Light Tests:

NOTE: Command line output from these can be found in the svm\_commands.txt and svm\_pca\_commands.txt files.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SVM-Light** | **768 Features** | | | **PCA Reduced to 17 Features** | | |
| **Kernel Mode** | **Accuracy** | **Precision** | **Recall** | **Accuracy** | **Precision** | **Recall** |
| 0: linear (Default) [poly degree 1] | 85.83% | 95.05% | 75.82% | 84.92% | 89.64% | 76.12% |
| 1: polynomial [degree 2] | 49.63% | Error | 0% | 53.85% | Error | 0% |
| 1: polynomial [degree 3] | 49.63% | Error | 0% | 53.85% | Error | 0% |
| 1: polynomial [degree 4] | 49.63% | Error | 0% | 53.85% | Error | 0% |
| 1: polynomial [degree 5] | 49.63% | Error | 0% | 53.85% | Error | 0% |
| 2: radial | 49.96% | 100% | 0.66% | 54.27% | 100% | 0.90% |
| 3: sigmoid tanh | 49.63% | Error | 0% | 55.51% | 51.66% | 56.01% |

# PCA analysis thoughts:

During the PCA analysis It came to my understanding that we don’t simply drop columns from the dataset to reduce the feature space, instead, we use the PCA process to “project” the higher dimensional feature-space onto a lower dimension. In class this is what you were hinting at but I did not understand at first. A 2D analysis showed that the 2nd dimension had little to know variance and thus, realistically, you could use the projected and remaining dimension by itself to classify.

Thus because the process projects and reduces the features, there is no way to determine which columns/features from the original dataset are actually still being used and most important.

It was awesome to see the features go from 768 down to 17, and only lose 1% accuracy.

## Example dataset thought experiment on opposite of PCA reduction:

One thought I had and have been pondering is. What if a large variance doesn’t actually indicate the separation of two datasets? What if the distinction is actually in the smaller variance dataset? Let’s say you had a dataset that had a feature column that only had a variance of 10 between its smallest value and its largest value. At first this would seem to be useless, but what if this feature column was in fact almost like a 1-1 correlation to the Y values?

Let’s assume a dataset where it has N features that are extremely variable (say a difference of 1000 minimum). Then we have 1-2 features that are only a difference of 10 max. Now, let’s assume that one of those two features is favorite color between 10 most common colors. If you are trying to group someone by their gender, the color MIGHT be pretty significant, but your PCA process would remove it because it doesn’t have a high amount of variance. I imagine that could be a problem in using PCA on some data sets.